SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Stručni studij elektrotehnike, smjer Informatika

WEB APLIKACIJA ZA UPRAVLJANJE RESTORANOM BRZE HRANE

Završni rad

Ante Tolušić

Osijek, 2018.

Sadržaj

1. UVOD

Tema završnog rada je izrada web aplikacije za upravljanje restoranom brze prehrane. Aplikacija omogućava vlasnicima restorana sastavljanje jelovnika i lakše upravljanje narudžbama. Korisnici upotrebom aplikacije mogu sami sastavljati željene sendviče. Osim predloženih menija, moguće je odabrati željene proizvode i sastavljati vlastite menije, te putem jednostavnog sučelja potvrditi narudžbu.

Glavni alat za izradu aplikacije je ReactJs biblioteka odnosno njegov ekosustav koji se temelji na Javascript programskom jeziku. Serverski dio aplikacije temelji se na Google-ovom Firebase-u.

Prije svega, dan je teorijski osvrt na korištene alate, tj. tehnologije prilikom izrade aplikacije. Nastavno tome, dan je vizualni opis nastanka aplikacije i kod na kojemu se zasniva web aplikacija.

1.1. Zadatak završnog rada

Zadatak ovog rada je razviti web aplikaciju koja će omogućiti vlasniku jednostavno vođenje restorana brzom hranom. Također, aplikacija sadrži i dio putem kojeg korisnici mogu naručivati hranu i sastavljati vlastite sendviče.

2. TEORIJSKI UVOD

U ovom dijelu završnog rada nalazi se teorijski uvid u tehnologije korištene za izradu web aplikacije.

2.1. JavaScript

JavaScript (kratica JS) je dinamični skriptni programski jezik najviše poznat po tome da se koristi za web stranice. Baziran je na prototipu, te podržava objektno orijentirani stil programiranja. Isto tako, koriste ga i okruženja izvan browsera poput node.js. U JS-u postoji dvije različite vrste podataka: objekti i primitivne vrste podataka. U primitivne vrste spadaju: undefined, string, null, number, boolean [1].

JS se izvršava na korisničkom dijelu weba i može se koristiti za određivanje ponašanja web stranica ovisno o „eventu“. JavaScript nije nastao kao nasljednik Jave, nego je ime dobio na temelju njezine popularnosti. Sintaksa JS-a je namjerno slična Javi i C++ kako bi se novi programski jezik lakše savladao.

JavaScript je najpopularniji skriptni jezik kojeg podržavaju gotovo svi web preglednici. Glavni je alat za izradu ove aplikacije.

2.2. HTML

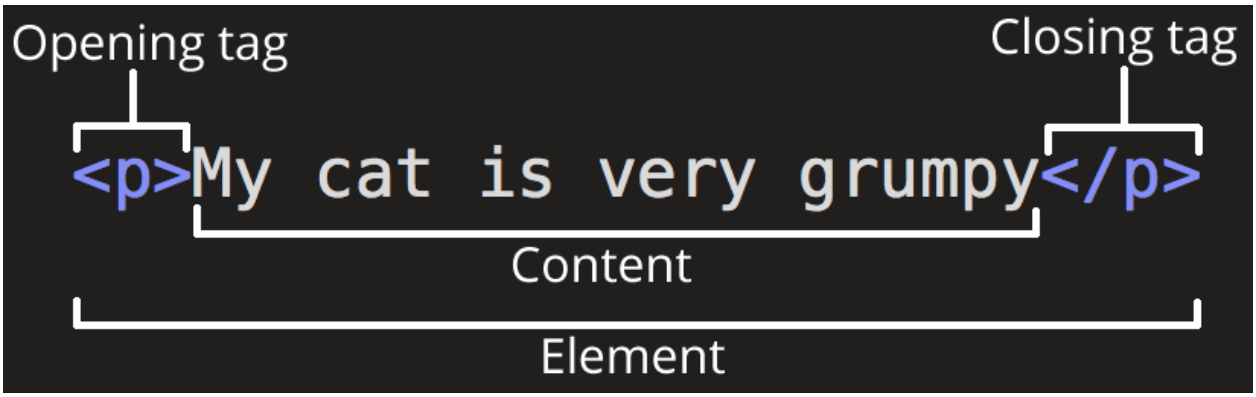
HTML (engl. *Hyper Text Markup Language*) je jezik koji se koristi za oblikovanje sadržaja web stranica. Nije programski nego je prezentacijski jezik što znači da se njime ne izvršavaju nikakvi zadatci nego opisuje sadržaj web stranice. HTML se sastoji od grupe elemenata, a neki od njih su: <html>, <link>, <button>, <body> koje možemo samozatvoriti ili „zamotati“ neki sadržaj unutar njih. Samozatvarajuće oznake mogu povećati ili smanjiti font, preusmjeriti korisnika na neku drugu stranicu itd. Prvi javno dostupan opis HTML-a je dokument pod nazivom HTML tags krajem 1991. godine. Sastojao se od dvadeset elemenata početnog, jednostavnog dizajna HTML-a. Prva verzija HTML jezika objavljena je 1993. godine [2].

Glavni zadatak HTML jezika je uputiti web preglednik kako prikazati hipertekst dokument pri čemu se nastoji da taj dokument izgleda jednako bez obzira o kojem je web pregledniku, računalu ili operacijskom sustavu riječ [2].

Glavni dijelovi elemenata su otvarajuće i zatvarajuće oznake, sadržaj i element.

- Otvarajuće oznake sastoje se od imena elementa, „zamotanog“ u otvarajuće „<“ i zatvarajuće „>“ zagrade. Otvarajuća oznaka je mjesto gdje započinje element;
- Zatvarajuće oznake su isto što i otvarajuće, samo sadrže „/“ prije imena elementa. Element završava zatvarajućim oznakama. Zagrade koje se koriste su „</“ i „>“;
- Sadržaj elementa i
- Element sačinjavaju otvarajuće oznake, zatvarajuće oznake i sadržaj [3].

Na Slici 2.1. prikazani su glavni dijelovi elemenata.



Slika 2.1. Glavni dijelovi elemenata [3].

Elementi također mogu imati i atribute. Atributi sadrže dodatne informacije o elementu koje zapravo nisu vidljive u sadržaju. Jedan od atributa je class pomoću kojeg više elemenata može dobiti npr. jednaku veličinu fonta, istu boju, odnosno bilo koji stil. Slika 2.2. daje prikaz atributa.

Slika 2.2. Prikaz elementa s class atributom [3].

2.3. CSS

CSS je skraćenica od “*Cascading Style Sheets*” i služi za definiranje stilova koji određuju izgled HTML elemenata.

Stilovi se nadovezuju u "*Style Sheets*", eksterne datoteke sa *.css* ekstenzijom ili se jednostavno pišu u zaglavlju HTML dokumenta, pa čak i na samim elementima. CSS je kreiran od strane W3C-a, a pojavio se s HTML-om 4.0 kao rješenje za veće potrebe odvajanja sadržaja stranice od dizajna. CSS omogućuje developerima kontroliranje stilova i izgled više HTML stranica istovremeno, tj. jedan stil za neki element može se iskoristiti na više stranica [4].

2.4. ReactJs

React je biblioteka za pravljenje korisničkih sučelja. Razvijen je i korišten od strane popularne društvene mreže Facebook koji ga je besplatno poklonio korisnicima. Iako ga mnogi nazivaju *frameworkom*, ReactJs je samo biblioteka jer je za izradu potpunog rješenja potrebno koristiti i ostale biblioteke i alate u kombinaciji s njim.

Svaka React aplikacija se sastoji od komponenata koje su samostalne i ponovno upotrebljive cjeline koda. Komponente sadrže markup, logiku i stil. Komponente su zapravo funkcije koje primaju ulazne parametre, a kao izlaz vraćaju element koji opisuje što se treba prikazati na zaslonu, tj. HTML. Treba ih izrađivati tako da je svaka pojedina komponentra zadužena za samo jedan dio funkcionalnosti što će omogućiti njihovu ponovnu upotrebu.

U React-u, komponente se mogu definirati na dva načina:

- Kao klase:
  React komponente prilikom nasljeđivanja *Component* klase dobivaju mogućnost korištenja „*state*" objekta koji prilikom svake njegove promjene poziva određene metode koje su zadužene za prikazivanje *JavaScript Syntax Extension (JSX)* elemenata, odnosno manipulacija istih. Na Slici 2.3. prikazana je React komponenta definirana kao klasa.

```
class FoodMenu extends Component {
  render() {
    return (
      <ul className="foodMenu">
        <li>Sandwich</li>
        <li>Cevapi</li>
        <li>Hamburger</li>
        <li>Cheeseburger</li>
      </ul>
    )
  }
}
```

- ***Slika 2.3.*** React komponenta definirana kao klasa koja nasljeđuje *React.Component* klasu.

- Kao funkcije:

Slika 2.4. prikazuje React komponentu koja je definirana kao funkcija, te nema pristup *state* objektu i ostalim metodama koje dobiva React komponenta definirana kao klasa koja nasljeđuje *React.Component* klasu.

```
const foodMenu = () => {
  return (
    <div>Menu</div>
  );
};
```

Slika 2.4. React komponenta definirana kao funkcija čiji je jedini zadatak vratiti *JSX* element.

React komponenta ovisi o dva tipa podataka: *state* i *props* objekti. *Parent* komponenta prosljeđuje *props* objekt koji je za *child* komponentu stalan kroz cijeli njezin životni ciklus. *State* objekt koristi se za promjenjive podatke, privatan je za komponentu i može se ažurirati uz pomoć *setState()* funkcije.

2.4.1. JSX

U prethodno navedenim primjerima vidljivo je da metoda *render()* ne vraća klasični JavaScript, već *JSX* koji je sličan HTML-u. *JSX* potječe od izraza *JavaScript Syntax Extension* koji je također napravio Facebook. *JSX* nije ništa više nego JavaScript funkcija, kao što je i vidljivo na Slikama 2.5. i 2.6..

```
<MyButton color="blue" shadowSize={2}>
  Click Me
</MyButton>
```

Slika 2.5. JSX element.

```
React.createElement(
  MyButton,
  {color: 'blue', shadowSize: 2},
  'Click Me'
)
```

Slika 2.6. Obična funkcija čiji je kod ekvivalentan Slici 2.5.

2.5. Firebase

Firebase je *Backend-as-a-Service(BaaS).* To je platforma za razvoj mobilnih i web aplikacija koja pruža potpuno rješenje bez kreiranja vlastitih servera, API-a (aplikacijsko programsko

sučelje) i baza podataka, koju korisnik može prilagoditi svojim potrebama. Firebase pruža mnoštvo servisa koji su dostupni korisniku, primjerice:

- *Realtime* baza podataka – većina baza podataka zahtjeva HTTP poziv za primanje podataka i vraćaju podatke samo kada ih se pita. Spajanje na Firebase bazu podataka ne obavlja se preko HTTP nego *WebSocketa* koji su puno brži nego HTTP.
- *File storage* – pruža jednostavan način za spremanje podataka, najčešće slika. Isto tako, posjeduje određenu zaštitu za autorizirane klijente.
- *Autentikacija* – Firebase posjeduje ugrađeno email/password autentikacijsko rješenje, te podržava Oauth2 za Google, Facebook, Twitter i GitHub.

2.5.1. Realtime baza podataka

Firebase Database je *NoSQL* baza podataka gdje se podatci spremaju u *JSON* formatu. Iz tog razloga cijelu bazu podataka možemo promatrati kao jedno *JSON* stablo u oblaku, te dodavanjem novog podatka u bazu, dolazi do stvaranja novog čvora u stablu s određenim ključem. Na Slici 2.7. dan je primjer stabla. Jezik Firebase baze podataka temeljen je na izrazima i vrlo je fleksibilan, te se koristi za definiranje pravila pristupa bazi i za definiranje strukture baze podataka.

Slika 2.7. Primjer stabla u Firebase bazi podataka [5].

2.5.2. Sigurnost Firebase Realtime baze podataka

Firebase daje jezik kojim se definiraju pravila pristupa pojedinim dijelovima stabla baze podataka. Ukoliko se korisniku da pravo pristupa određenom čvoru, on istovremeno dobiva pravo pristupa za cijelo stablo koje potječe iz tog čvora.

3. WEB APLIKACIJA ZA VOĐENJE RESTORANA BRZE PREHRANE

3.1. Ideja

Ideja ove aplikacije je olakšati komunikaciju između vlasnika restorana i kupca, te pojednostaviti, ubrzati i pružiti što kvalitetniju uslugu.

Ulaskom u web aplikaciju, korisniku se prikazuju dostupni jelovnici kojima upravlja vlasnik restorana. Vlasnik restorana ima mogućnost raditi promjene na jelovnicima odnosno dodavati ih, uklanjati, uređivati te određivati njihovu cijenu uz prije obavljenu autentikaciju. Osim toga, vlasnik može upravljati narudžbama i pratiti učestalost naručenih obroka uz pomoć grafičkog prikaza. Radi boljeg korisničkog iskustva, vlasnik može postavljati fotografije predloženih jelovnika, te ih mijenjati ovisno o promjenama i akcijama restorana. Isto tako, korisnik može sam sastavljati narudžbu unutar koje ima dodatnu mogućnost sastavljanja sendviča po želji. Prilikom sastavljanja sendviča korisnik može odabrati jedan ili više ponuđenih sastojaka uz koje je navedena i cijena. Korisnik ne može potvrditi narudžbu prije unosa osobnih podataka.

3.2. Aplikacija

Kod aplikacije se može pronaći na linku:

https://github.com/tole9/react-restaurant-management

https://tole9.github.io/react-restaurant-management/#/

Svaka React aplikacija ima svoja glavna obilježja, te datoteke od kojih se sastoji. Za jednostavniju izradu ove aplikacije i upravljanje datotekama korišten je alat *Create React App* (CRA) pomoću kojeg je sačuvano vrijeme i olakšano postavljanje konfiguracije projekta, te upravljanje datotekama. Prije svega potrebno je instalirati node.js koji omogućuje korištenje njegovog *package managera (npm).* Nakon toga slijedi instalacija CRA uz pomoć npm-a unosom sljedeće naredbe u komandnu liniju:

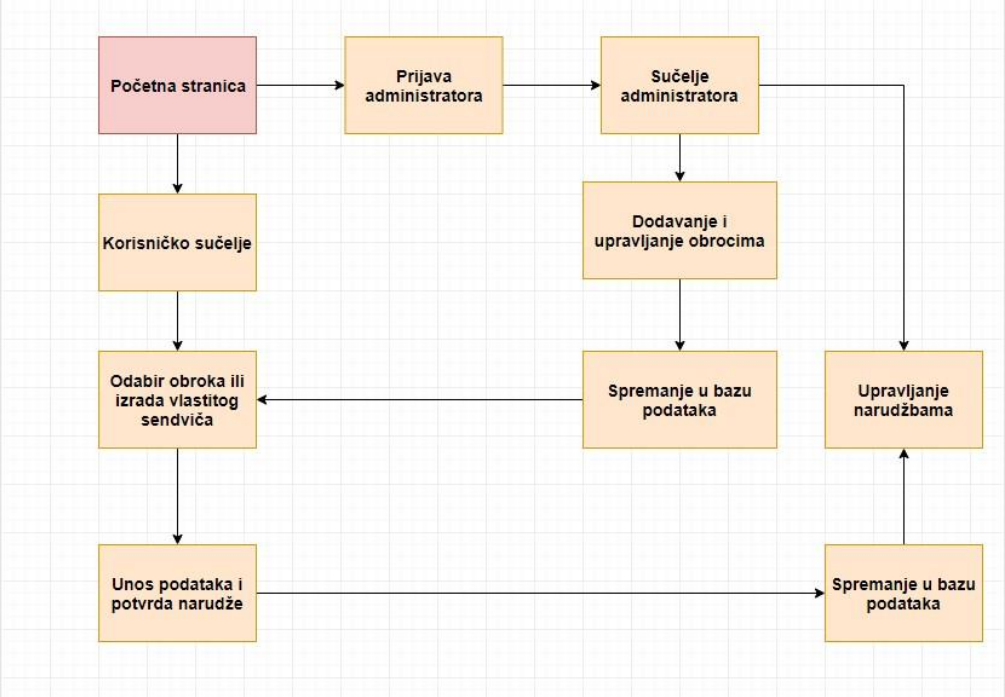
```
npm init react-app my-app
```

Također, uz CRA se dobiva i *development server* koji olakšava razvoj aplikacije bez dodatnog osvježavanja preglednika pri izmjeni određenih dijelova koda. Za pregled aplikacije na *development serveru* potrebno je upisati iduću naredbu u komandnu liniju:

```
npm start
```

3.2.1. Model aplikacije

Model ove aplikacije prikazan je u obliku dijagrama gdje je dan slijed odvijanja aplikacije. Dijagram je prikazan na Slici 3.1. Početna stranica daje izbor pristupu korisničkom ili administrativnom sučelju uz obaveznu prethodnu prijavu u sustav putem odgovarajućih podataka.



Slika 3.1. Slijedni prikaz aplikacije.

Ukoliko je prijava administratora uspješna, otvara se sučelje gdje vlasnik restorana može dodavati i uređivati obroke, omogućavati posebne ponude i akcijske cijene. Prilikom uređivanja obroka moguće je dodavati fotografije i opise pojedinih jela za vjerniji prikaz. Posebnost ove aplikacije je mogućnost izrade vlastitih sendviča. Vlasnik restorana određuje hoće li omogućiti korisniku izradu sendviča, tj. koje sastojke može koristiti za izradu sendviča. Svaka zaprimljena narudžba, tj. detalji se prate u posebnom dijelu za narudžbe gdje je moguće označiti njihov status

ovisno o isporučenosti. Isto tako, u dijelu s narudžbama nalazi se i dijagram koji vjerno prikazuje učestalost naručivanja pojedinih obroka.

Ulaskom u korisničko sučelje, korisnik dobiva mogućnost odabira već gotovih obroka ili pripravljanja vlastitih sendviča. Kao što je već prethodno spomenuto, sendvič je moguće napraviti samo s trenutno dostupnim sastojcima. Korisničko sučelje ima i košaricu u kojoj su vidljivi svi obroci koje je korisnik prethodno odabrao. Potvrda narudžbe zahtjeva unos osobnih podataka.

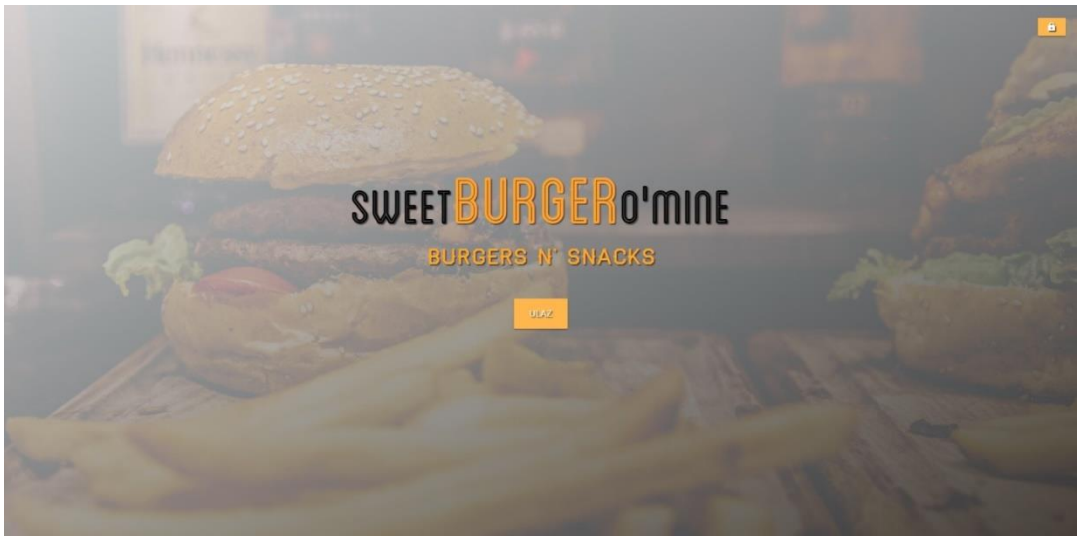
3.2.2. Root aplikacije

U kodu vidljivom ispod nalazi se React komponenta sa *render()* metodom koju imaju samo komponente koje nasljeđuju *React.Component* klasu, a ona vraća *JSX* elemente odnosno elemente vidljive na sučelju. *Router* komponenta uvezena iz *react-router-dom* biblioteke koristi HTML5 history API i služi za sinkronizaciju korisničkog sučelja sa URL-om. Prilikom izmjene URL-a neće doći do osvježavanja preglednika, a bit će prikazana komponenta ovisno o zatraženoj putanji. U *Route* komponentama definiramo *path string* koji je putanja na kojoj se prikazuje određena komponenta. *Path*, *component* i *exact* su dio *props* objekta kojemu se može pristupiti unutar *Route* komponente. *Exact* svojstvo nema pridruženu vrijednost i zadana vrijednost mu je true, a on označava da za točno navedenu putanju u *path stringu* prikaže komponentu pridruženu *component* svojstvu.

```
class App extends Component {
  render() {
    return (
      <Router>
        <div className="App">
          <Switch>
            <Route exact path="/" component={LandingPage} />
            <Route exact path="/menu" component={Menu} />
            <Route path="/menu/checkout" component={Checkout} />
            <Layout>
              <Route exact path="/adminpanel" component={AdminPanel} />
              <Route exact path="/adminpanel/orders" component={OrderStats} />
              <Route path="/meals/:id" component={MealDetails} />
            </Layout>
          </Switch>
        </div>
      </Router>
    );
  }
}
```

3.2.3. Početna stranica

Prilikom pokretanja aplikacije u pregledniku, otvara se početna stranica prikazana na Slici 3.2. Na stranici se nalaze dva gumba. Pritiskom gumba na sredini stranice, otvara se korisničko sučelje aplikacije. S druge strane, pritiskom na gumb u gornjem desnom kutu, otvara se forma za prijavu administratora.



Slika 3.2. Početna stranica web aplikacije.

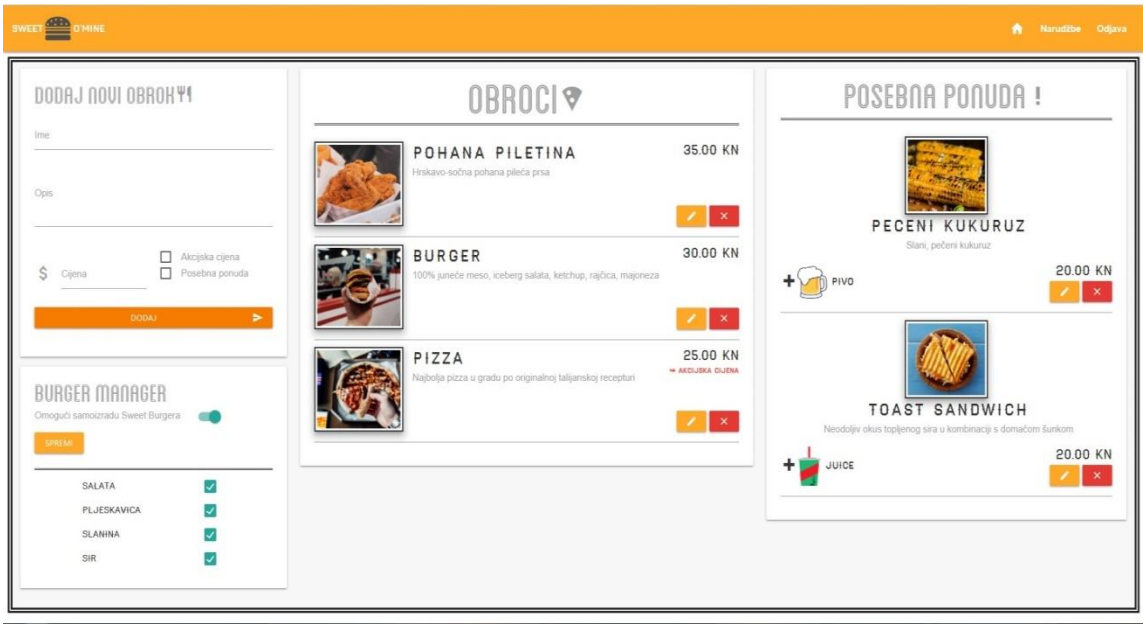
Za prikazivanje početne stranice, odnosno React komponente korištena je JavaScript funkcija koja samo vraća *JSX* element, a ne klasa jer nije potreban „*state*“ objekt i metode koje naslijedi klasa. Kao što je vidljivo u kodu ispod funkcija *LandingPage* vraća samo jedan element jer React zahtjeva da svi elementi komponente budu zatvoreni unutar jednog elementa. Iz priloženog koda se može vidjeti razlika u pisanju CSS klasa da je ime atributa *className* umjesto *class* jer kao što je već navedeno *JSX* elementi su obične funkcije koje zahtjevaju određene parametre, a *class* je rezervirana ključna riječ u JavaScript-u. *Link* element je komponenta uvezena iz *react-router-dom* biblioteke koja klikom na nju prikazuje drugu komponentu ovisno o putanji, tj. URL-u bez dodatnog osvježavanja preglednika.

```
const LandingPage = () => {
  return (
    <div className={classes.LandingPage}>
      <div className={classes.MoveUp}>
        <Link
          to="/adminpanel"
          className={`btn-small orange lighten-2 ${classes.AdmBtn}`}
        >
          <i className="material-icons">lock</i>
        </Link>
        <span>sweet</span>
        <span className={classes.BurgerTitle}>Burger</span>
        <span>o'mine</span>
        <p>Burgers N' Snacks</p>
```

```
        <div className={classes.LPBtns}>
          <Link
            to="/menu"
            className="btn-large waves-effect waves-light orange lighten-2"
          >
            Ulaz
          </Link>
        </div>
      </div>
    </div>
  );
};
```

3.2.4. Administratorski dio

Nakon uspješnog unosa korisničkog imena i lozinke, otvara se sučelje prikazano Slikom 3.3. koje je namijenjeno vlasniku restorana. Administrator u ovom dijelu dodaje obroke koji su raspoloživi za narudžbu i pohranjuje ih u Firebase bazu podataka. Svaki obrok može imati akcijsku cijenu ili biti uključen u posebnu ponudu kojoj se mogu dodati neki od raspoloživih priloga poput pića ili slastica. Za vjerniji prikaz jela, vlasnik ima mogućnost dodavanja fotografija koje se spremaju u Firebase storage. Osim toga, jedan od glavnih aduta ove aplikacije je i *Burger Manager* pomoću kojeg se kontroliraju sastojci trenutno dostupni korisniku za izradu vlastitog sendviča.



Slika 3.3. Sučelje namijenjeno vlasniku restorana

3.2.4.1. Forma za dodavanje i uređivanje obroka

Za dodavanje obroka definirana je komponenta *MealForm* koja nasljeđuje *React.Component* klasu što znači da je potreban *state* objekt za manipulaciju podacima. U ovom slučaju, *state* je zadužen za kontrolu polja u formi u kojoj se popunjavaju podaci vezani za obrok. Isto tako, važno je da se komponente pišu u određenoj formi kako bi se mogle ponovno iskoristiti. Ovdje se *MealForm* komponenta koristi za dodavanje i uređivanje obroka. Da bi se *MealForm* mogao koristiti u oba slučaja, iz *parent* komponente potrebno je proslijediti svojstvo koje će biti dostupno u *props* objektu unutar *MealForm*. Na temelju njega će se odrediti hoće li se pri potvrđivanju forme obaviti dodavanje ili ažuriranje obroka.

Potvrđivanje forme obavlja se klikom na gumb čiji tekst ovisi o *props* objektu, odnosno sadrži li *props* objekt *editMeal* svojstvo. Ako je *editMeal* svojstvo postojano, tekst će biti „Uredi“, a u suprotnom „Dodaj“. U kodu niže prikazana je metoda koja se poziva klikom na navedeni gumb. Također, slijed metode ovisi o svojstvu *editMeal*. Ako *props* objekt sadrži *editMeal*, poziva se metoda *update()* iz *firestore* objekta koji je uvezen iz *react-redux-firebase* biblioteke, a služi za integraciju React aplikacije s Firebase platformom. *update()* metoda prima ažurirane podatke, te *ID* obroka kojeg je potrebno izmijeniti. Nakon uspješnog zahtjeva poziva se *setState()* metoda koja će promijeniti dio *state* objekta i omogućiti pojavu određene poruke na sučelju.

```
onSubmit = e => {
  e.preventDefault();

  const { name, price, desc, specialOffer, discount, id, specialOfferItem, img } = this.state;
  const { firestore, editMeal } = this.props;

  const data = { name, price, desc, specialOffer, discount, specialOfferItem, img };

  if (editMeal) {
    if (!specialOffer) {
      this.setState({ specialOfferItem: null });
      data.specialOfferItem = null;
    }
    firestore.update({ collection: "meals", doc: id }, data).then(() => {
      this.setState({ successMsg: true });
      setTimeout(() => this.setState({ successMsg: false }), 2000);
    });
  } else {
    firestore.add({ collection: "meals" }, data);
    this.setState({
      name: "",
      price: "",
      desc: "",
      specialOffer: false,
      discount: false,
      specialOfferItem: null
    });
  }
};
```

U slučaju da svojstvo *editMeal* ne postoji, tj. da nije proslijeđeno od *parent* komponente metoda ulazi u *else* dio koda koji je vidljiv iznad. Poziva se *add()* metoda iz već spomenutog *firestore* objekta i izvršava se dodavanje obroka u bazu podataka uz pomoć Firebase API-a. Nakon uspješnog dodavanja obroka poziva se *setState()* i dolazi do promjene *state* objekta, odnosno postavljanja podataka vezanih za formu na početnu vrijednost (prazna polja forme).

3.2.4.2. Popis obroka

Funkcija *Meals* zadužena je za prikaz svih obroka. Navedena komponenta pri svom pozivu radi upit na kolekciju podataka pod nazivom „*meals*“ u Firebase bazi podataka uz pomoć *react-redux-firebase* biblioteke. S obzirom da je to asinkroni zadatak odnosno *JavaScript engine* nastavlja izvršavanje ostalog koda ne čekajući odgovor upita, za vrijeme trajanja tog zadatka bit će prikazana komponenta *Spinner*. Zadaća te komponente je da prikaže CSS animaciju sve dok ne stignu podaci sa servera. Kada zatražena kolekcija podataka bude dostupna u komponenti zamijenit će *Spinner*. Funkcija *Meals* prikazuje obroke u dvije odvojene sekcije ovisno o tome pripadaju li posebnoj ili standardnoj ponudi.

U kodu prikazanom niže vidimo da jedino što funkcija *Meals* vraća je varijabla *mealList*. Navedena varijabla za vrijeme upita ima pridruženu vrijednost, tj. komponentu *Spinner* sve dok podaci sa servera ne budu dostupni unutar tijela funkcije. Od podataka sa servera kreira se polje *JSX* elemenata koji će biti pridruženi *mealDetails-u*. Za svaki element polja instancira se *Meal* klasa s određenim svojstvima dostupnim u *props* objektu. *Meal* predstavlja pojedini obrok.

```
const Meals = props => {
  let mealList = (
    <div className={classes.SpinnerDiv}>
      <Spinner />
    </div>
  );
  const { meals, spec } = props;
  if (meals) {
    mealList = meals.map(meal => {
      if (spec && meal.specialOffer) {
        return (
          <li key={meal.id} className={classes.MealLi}>
            <Meal mealDetail={meal} spec={props.spec} />
          </li>
        );
      } else if (!spec && !meal.specialOffer) {
        return (
          <li key={meal.id} className={classes.MealLi}>
            <Meal mealDetail={meal} spec={props.spec} />
          </li>
        );
      } else {
        return null;
      }
    });
  }

  return mealList;
```

```
};

export default compose(
  firestoreConnect(["meals"]),
  connect((state, props) => ({
    meals: state.firestore.ordered.meals
  }))
)(ReactFontFace(Meals, fontSecondary));
```

3.2.4.3. Pojedini obrok

Za svaki obrok kreira se *Meal* komponenta koja prima svojstva putem *props* objekta iz *parent* komponente. Uz obroke se nalazi i fotografija koja omogućava korisniku lakši odabir zbog vjernijeg prikaza. Ukoliko administrator nije dodao istu, tj. ako svojstvo *img* iz *props* objekta nije proslijeđeno u navedenu komponentu, bit će prikazana fotografija koja ukazuje na to da u ovom trenutku prikladna fotografija nije postavljena. Isto tako, biti će prikazana UploadImage komponenta putem koje se može postaviti fotografija obroka vidljiva na Slici 3.4.

Slika 3.4. Obrok bez dodane prikladne fotografije.

State objekt *UploadImage* komponente sadrži svojstvo koje kontrolira mogućnost klika na gumb za dodavanje fotografije. Gumb će biti onemogućen sve dok nije odabrana fotografija. Klikom na gumb vidljiv na Slici 3.4., poziva se metoda *onUploadClickHandler()* koja kontrolira tijek ažuriranja fotografije. Poziva se funkcija *toggleLoading()* koja prikazuje *Spinner* komponentu za asinkroni zadatak ažuriranja fotografije. Metoda *put() storage* objekta uvezenog iz *react-redux-firebase* biblioteke sprema sliku u Firebase *storage* i vraća objekt iz kojeg možemo pristupiti URL-u spremljene fotografije. Nakon toga, poziva se *update()* metoda za pridruživanje URL-a fotografije određenom obroku, te se ponovno izvršava *toggleLoading()* za uklanjanje *Spinner* CSS animacije i na sučelju se prikazuje željena fotografija obroka.

Osim mogućnosti dodatka fotografije, *Meal* komponenta sadrži i dva gumba za pregled i uređivanje detalja obroka, te uklanjanje obroka iz baze podataka kao i sa sučelja. Prvi gumb je već spomenuta *Link* komponenta koja mijenja URL bez dodatnog osvježavanja stranice i

prikazuje detalje obroka koji se mogu naknadno uređivati. Putanji *Link* komponente prosljeđuje se *ID* obroka koji je vidljiv u URL-u. Pri pokretanju *MealDetails* komponente zadužene za prethodno navedene zadatke šalje se upit na Firebase bazu podataka s *ID* parametrom iz URL-a. Server odgovara s prikladnim podacima koji su izravno vezani s poslanim *ID*-om.

3.2.4.4. Burger Manager

Kao jedan od glavnih aduta ove aplikacije je *Burger Manager* putem kojeg vlasnik omogućava ili onemogućava korisniku izradu sendviča po želji. Također, moguće je označiti sastojke koji će biti dostupni korisniku za izradu sendviča, kao što su pljeskavica, salata, sir i drugi. Komponenta *BurgerIngredientsManager* koristi *state* objekt za manipulaciju podataka što znači da mora naslijediti *React.Component* klasu. Nakon prikaza navedene komponente, odnosno izvršavanja *render()* metode poziva se *componentDidMount()* metoda koja se okida zbog nasljeđivanja *React.Component* klase. Ta metoda izvršava se samo jednom i idealna je za upućivanje upita na server i pozivanja *setState()* metode za ažuriranje *state* objekta. Kod niže prikazuje *componentDidMount()* metodu unutar *BurgerIngredientsManager* komponente. *get()* metoda firestore objekta šalje upit na „*burger*“ kolekciju u Firebase bazi podataka, te s uspješno dohvaćenim podacima ažurira *state* objekt vidljiv u kodu ispod koji svojom promjenom prikazuje prethodno postavljena i u bazu unešena stanja sastojaka, tj. dozvolu izrade vlastitog sendviča.

```
state = {
  allowBuild: true,
  salata: true,
  pljeskavica: true,
  slanina: true,
  sir: true,
  successMsg: false
};

componentDidMount() {
  const { firestore } = this.props;
  firestore.get("burger/burger").then(data => {
    if (data.exists) {
      this.setState({ ...data.data() });
    }
  });
}
```

Klikom na gumb spremi vidljivom na Slici 3.3. poziva se metoda *onSaveClickHandler()* koja ažurira „*burger*“ kolekciju u Firebase bazi podataka s podatcima iz *state* objekta.

3.2.4.5. Zaprimljene narudžbe

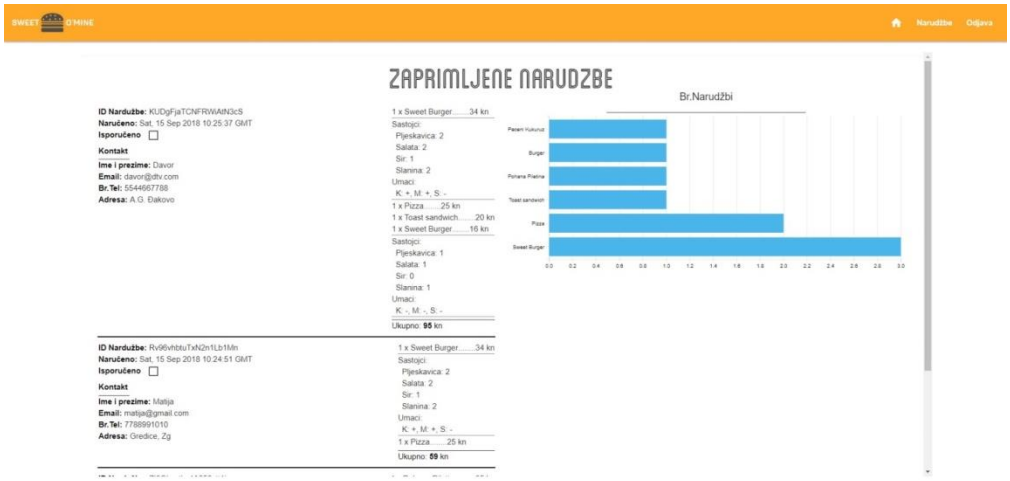
Klikom na *Link* komponentu „Narudžbe“ na navigacijskoj traci administratorskog sučelja prikazuje se *OrderStats* komponenta kojoj ne treba *state* objekt, te je definirana kao funkcija. *OrderStats* prikazuje listu svih zaprimljenih narudžbi naručenih putem korisničkog sučelja ove aplikacije. Uz pomoć *react-redux-firebase* biblioteke šalje se upit na server, tj. kolekciju podataka „*orders*“ u Firebase bazi. Firebase odgovara sa svim zaprimljenim narudžbama i šalje ih u komponentu u kojoj je zatražen upit. Firebase sve zaprimljene narudžbe šalje u polju elemenata. Svaki element polja je JavaScript objekt sa svojstvima kao što su *ID* narudžbe, popis naručenih obroka, vrijeme naručivanja, te kontakt podatci. Na navedeno polje pridošlo iz Firebase baze podataka poziva se JavaScript *Array* pomoćna metoda *map()* koja prolazi kroz to polje i za svaki element vraća stukturu *JSX* elemenata, odnosno jednog *JSX* elementa vezanog za tu narudžbu. Također, uz svaku narudžbu se prikazuje i „*checkbox*“ koji je moguće označiti ako je narudžba isporučena. Osim liste svih narudžbi, ova komponenta daje i grafički prikaz učestalosti naručenih obroka. Dijagram korišten za grafički prikaz uvezen je iz biblioteke *react-charts* koji zahtjeva da mu se u *props* objekt proslijedi određena struktura podataka kako bi se graf mogao iscrtati. U kodu vidljivom niže razrađena je potrebna stuktura koju dijagram zahtjeva.

```
let chartData = [];

orders.forEach(order => {
  order.orders.forEach(o => {
    if (chartData.length > 0) {
      let pushed = false;
      chartData.forEach(cd => {
        if (cd.x === o.name) {
          cd.y++;
          pushed = true;
        }
      });
      if (!pushed) {
        chartData.push({ x: o.name, y: 1 });
      }
    } else {
      chartData.push({ x: o.name, y: 1 });
    }
  });
});

let chart = (
  <Chart
    data={[{ label: "Br.Narudzbi",data: chartData }]}
  >
    <Axis primary type="ordinal" position="left" />
    <Axis type="linear" stacked position="bottom" />
    <Series type={Bar} />
    <Cursor primary />
    <Cursor />
    <Tooltip />
  </Chart>
);
```

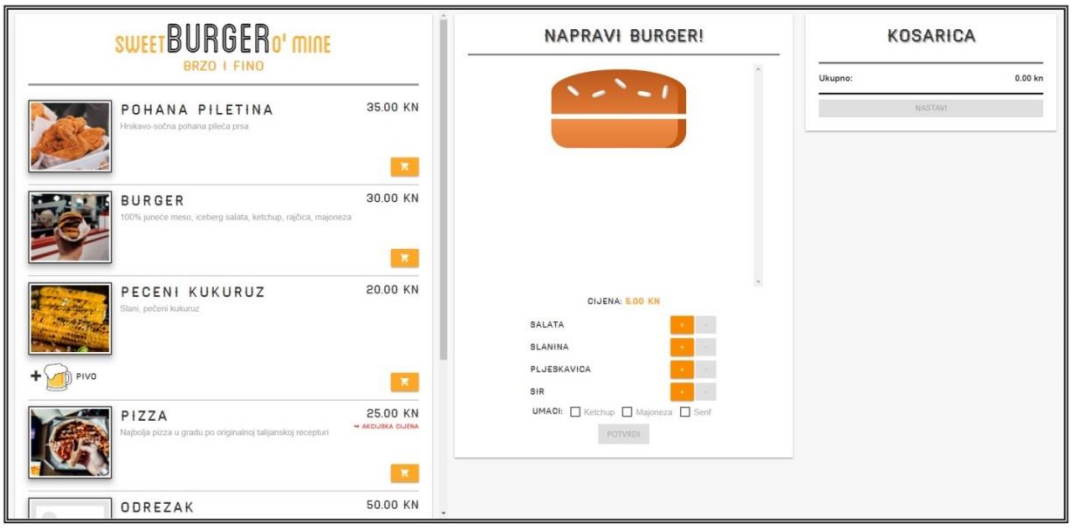
Definirana je varijabla *chartData* kojoj je pridruženo prazno polje koje će biti ispunjeno s potrebnom stukturom. Dijagram će se iscrtati samo ako putem *props* objekta primi polje JavaScript objekata koji za ključ „x“, tj. njegovu pridruženu vrijednost zahtjeva ime obroka, a za ključ „y“ broj učestalosti naručivanja tog obroka. Za razliku od navedene JavaScript *Array* pomoćne metode *map(), forEach()* metoda samo prolazi kroz polje elemenata obavljajući zadane operacije bez kreiranja novog polja. Na Slici 3.5. nalazi se *OrderStats* komponenta koja prikazuje zaprimljene narudžbe i s iscrtanim dijagramom koji vjerno prikazuje učestalost naručivanja određenih obroka.



Slika 3.5. Prikaz *OrderStats* komponente.

3.2.5. Korisnički dio

Klikom na *Link* komponentu na početnoj stanici aplikacije otvara se sučelje aplikacije namijenjeno korisniku vidljivo na Slici 3.6.. Putem ovog sučelja korisnik može odabrati neke od obroka koje je prethodno dodao vlasnik. Također, moguće je i izraditi vlastiti sendvič ako je vlasnik omogućio tu opciju. Navedene opcije implementirane su u jednostavno i interaktivno sučelje. S desne strane sučelja nalazi se košarica u kojoj su vidljivi svi odabrani obroci, tj. sendviči. U slučaju da je korisnik promijenio mišljenje nakon dodanog obroka ima mogućnost uklanjanja istog iz košarice.



Slika 3.6. Korisničko sučelje.

3.2.5.1. Lista prethodno dodanih obroka/Meni

S lijeve strane korisničkog sučelja nalazi se lista obroka koje je prethodno dodao vlasnik ili Meni. U njemu se nalaze obroci prisutni u standardnoj ponudi restorana, obroci na akcijskoj cijeni ili pak obroci u posebnoj ponudi gdje su uključeni dodatni prilozi po jedinstvenoj cijeni. Ima li neko jelo akcijsku cijenu, naznačeno je ispod cijene. Ako je neko jelo uključeno u posebnu ponudu, ispod fotografije se nalazi slikovni prikaz priloga. Kako bi prikaz jela bio što vjerniji, vlasnik je dodao i fotografije, te opise.

Ova komponenta veoma je slična komponenti zaduženoj za upravljanje listom obroka u administracijskom sučelju. Upravo to je i razlog zašto se React koristio u ovom projektu jer je ponovno korištenje već napisanih komponenata vrlo jednostavno. Sve što se treba napraviti je iz *parent* komponente proslijediti određeni parametar koji će biti pohranjen u *props* objekt na temelju kojeg će se prikazati određeni *JSX* elementi esencijalni za zamišljenu funkciju.

Kada usporedimo Sliku 3.6. i Sliku 3.3., vidljivo je da je jedina razlika među njima u gumbovima za interakciju. Također, razlika je i u tome što se neće pojaviti mogućnost ažuriranja fotografije ukoliko prethodno fotografija nije postavljena. To je sve moguće već prethodno spomenutim prosljeđivanjem određenog parametra *props* objektu.

3.2.5.2. Izrada vlastitog sendviča

Dizajn kojim je prikazan sendvič izrađen je uz pomoć https://www.youtube.com/watch?v=Lm_swJhK7Xw.

Između već spomenutih opcija korisničkog sučelja, tj. liste obroka i košarice, nalazi se bitan adut ove aplikacije, odnosno restorana, a to je odjeljak koji omogućava korisniku izradu vlastitih sendviča. Korisnik je u mogućnosti izabrati onu količinu sastojaka koju želi, te istovremeno sve umake koji su mu na raspolaganju. Vlasnik restorana u administratorskom sučelju određuje koji će sastojci korisniku biti na raspolaganju za izradu vlastitog sendviča ovisno o stanju istih na skladištu.

Komponenta *BurgerBuilder*, koja je zadužena za gore navedene operacije, pri svom pokretanju radi upit na kolekciju podataka „burger“ u Firebase bazi podataka uz pomoć već spomenute biblioteke *react-redux-firebase*. Podaci s kojim Firebase odgovara su prethodno postavljena svojstva od administratora vezana za izradu burgera. Odgovor Firebase-a se sprema u svojstvo *burgerStatus* u *props* objekt. U ovisnosti o vrijednosti *burgerStatus* svojstva, bit će prikazana već spomenuta *Spinner* komponenta, odnosno cijeli sustav za izradu sendviča. Ispod je vidljiv kod koji se izvršava ako su dostupni podatci iz upita.

```
    const { burgerStatus } = this.props;

    if (burgerStatus) {
      const disableIngs = {
        pljeskavica: burgerStatus.burger.pljeskavica,
        salata: burgerStatus.burger.salata,
        sir: burgerStatus.burger.sir,
        slanina: burgerStatus.burger.slanina
      };
      renderBurger = (
        <div className={classes.BurgerBuilder}>
          <Burger
            disableBurger={burgerStatus.burger.allowBuild}
            ings={{ ...this.state.ingredients }}
          />
          <BuildControls
            disableIngs={disableIngs}
            disableControls={burgerStatus.burger.allowBuild}
            disableBtn={disable}
            getPrice={this.getPrice}
            onAddIngHandler={this.onAddIngHandler}
            onDeleteIngHandler={this.onDeleteIngHandler}
            ings={{ ...this.state.ingredients }}
          />
        </div>
      );
    }

    return renderBurger;
  }
}

export default compose(
  firestoreConnect(["burger"]),
  connect(state => ({
```

```
    burgerStatus: state.firestore.data.burger
  }))
)(BurgerBuilder);
```

Na temelju podataka iz *burgerStatus* objekta izrada burgera će biti omogućena ili onemogućena. Komponenta *BurgerBuilder* se sastoji od dvije *child* komponente *Burger* koja simulira izgled hamburgera, tj. dodane sastojke koji se kontroliraju drugom *child* komponentom, *BuildControls*. *BurgerBuilder* koristi *state* objekt za povezivanje te dvije *child* komponente. Metodama *onIngAddHandler()* i *onDeleteHandler()* koje se prosljeđuju *BuildControls* komponenti, ažurira se *state* objekt *BurgerBuildera* koji se prosljeđuje *Burger* komponenti i ona prikazuje CSS stilizirane *JSX* elemente, te se na korisničkom sučelju dobiva vjerni prikaz sendviča, tj. sastojaka koji su dodani. Ispod je vidljiv kod za dvije gore navedene metode koje ažuriraju *state* objekt. Logika u obje metode je veoma slična, odnosno jedina razlika je što *onIngAddHandler()* metoda dodaje sastojak, dok ga druga uklanja.

```
  onAddIngHandler = ing => {
    this.setState({
      ...this.state,
      ingredients: {
        ...this.state.ingredients,
        [ing]: this.state.ingredients[ing] + 1
      }
    });
  };

  onDeleteIngHandler = ing => {
    this.setState({
      ...this.state,
      ingredients: {
        ...this.state.ingredients,
        [ing]: this.state.ingredients[ing] - 1
      }
    });
  };
```

BuildControls komponenti prosljeđuje se varijabla *disable* koja ima vrijednost *true* ili *false*. Ta vrijednost se dobiva tako što se koristi JavaScript *Array* pomoćna metoda *every()* koja vraća jednu od te dvije vrijednosti. Navedena metoda prolazi kroz polje elemenata sastavljeno od *state* objekta *BurgerBuilder* komponente i provjerava imaju li svi elementi veću vrijednost od nule. Uz pomoć *disable* svojstva koje je dostupno u *props* objektu *BuildControls* komponente onemogućava se dodavanje sendviča u košaricu ukoliko nije dodan barem jedan sastojak u sendvič.

BuildControls komponenta također koristi *state* objekt za praćenje cijene sendviča koja ovisi o dodanim sastojcima. Svaki od gumbova vidljivih u kodu niže vezanih za dio korisničkog sučelja zaduženog za dodavanje i uklanjanje sastojaka prilikom izrade vlastitih sendviča u svom *onClick* osluškivaču poziva dvije metode. Jedna je *onIngAddHandler()* ili *onDeleteHandler()*

ovisno o kliknutom gumbu, a druga *updatePrice()* koja ažurira *state* objekt što znači da svakim klikom na određeni gumb na sučelju je prikazana trenutna cijena sendviča. Klikom na gumb „Potvrdi“ poziva se metoda *BuildControls* komponente *onBurgerSubmitHandler()* kojom se prvi put koristi globalni *state* objekt koji je dostupan u cijeloj aplikaciji i nije striktno vezan za jednu komponentu. Korištenje globalnog *state* objekta omogućeno je uvozom biblioteke *react-redux*. Niže je vidljiv kod za *onBurgerSubmitHandler()* metodu.

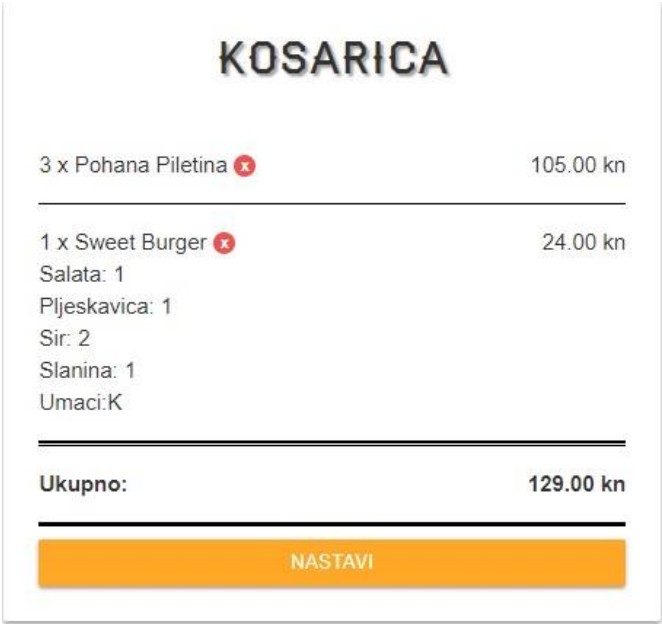
```
onBurgerSubmitHandler = () => {
  const { price, sauces } = this.state;
  const { ings, addMealToOrders } = this.props;
  const id = {
    salata: ings.salata,
    slanina: ings.slanina,
    pljeskavica: ings.pljeskavica,
    sir: ings.sir,
    majoneza: sauces.mayo,
    ketchup: sauces.ketchup,
    senf: sauces.mustard
  };
  let payload = {
    price,
    ings,
    name: "Sweet Burger",
    // kreiranje svog id-a radi counta
    id: JSON.stringify(id),
    count: 1,
    sauces: {
      ...sauces
    }
  };
  addMealToOrders(payload);
};
```

Svaki od obroka, uključujući i sendvič koji se izrađuje, mora imati svoj *ID* na osnovu kojeg se u košarici računa broj istih obroka. *ID* za sendvič implementiran je na taj način tako što se napravi JavaScript objekt od svih sastojaka izrađenog sendviča i njihove pridružene količine. Objekt se uz pomoć JavaScript metode *JSON.stingify()* pretvara u *string*, te on predstavlja *ID* za taj sendvič što znači da ako u trenutnoj narudžbi postoje dva izrađena sendviča s istim sastojcima u košarici će biti vidljiv jedan obrok s naznakom da je dodan dva puta. Pozivanjem *addMealToOrders()* metode vidljive u kodu iznad narudžba, odnosno sendvič se dodaje u globalni *state* objekt.

3.2.5.3. Košarica

S desne strane korisničkog sučelja vidljivog na Slici 3.6. nalazi se košarica koja prikazuje listu odabranih obroka. Komponenta *Orders* zadužena je za prikaz košarice. Uz pomoć *connect()* metode komponenta je u mogućnosti koristiti, tj. čitati podatke iz globalnog *state* objekta.

U globalnom *state* objektu definirano je polje elemenata u koje se dodaju obroci interakcijom gumbova na popisu obroka ili pri izgradnji vlastitog sendviča. Na temelju duljine tog polja omogućava se ili onemogućava gumb „Nastavi“, odnosno *Link* komponenta koja mijenja URL i prikazuje *Checkout* komponentu. To znači da će gumb biti onemogućen sve dok se ne doda barem jedan obrok u košaricu, tj. u polje globalnog *state*-a. Svaki obrok u košarici bit će prikazan s odgovarajućom cijenom, količinom dodanih istih obroka, te gumbom za uklanjanje neželjenog obroka iz košarice što je prikazano na Slici 3.7. Osim toga, vidljiva je i ukupna cijena svih dodanih obroka.



Slika 3.7. Primjer dodanih obroka u košaricu.

Klikom na „x“ gumb vidljiv na slici iznad poziva se metoda koja ažurira globalni *state* objekt tako što prihvaća proslijeđeni *ID* i poziva JavaScript *Array* pomoćnu metodu *filter()* nad poljem narudžbi u globalnom *state* objektu koja vraća novo polje elemenata bez obroka na koji je kliknut gumb.

3.2.5.4. Detalji kupovine

Kada korisnik odabere sve željene obroke i u košarici klikne na gumb „Nastavi“, otvaraju se detalji kupovine, tj. *Checkout* komponenta gdje su vidljivi obroci dodani u košaricu, ukupna cijena, te forma koju je potrebno popuniti za potvrđivanje narudžbe. *Checkout* komponenta

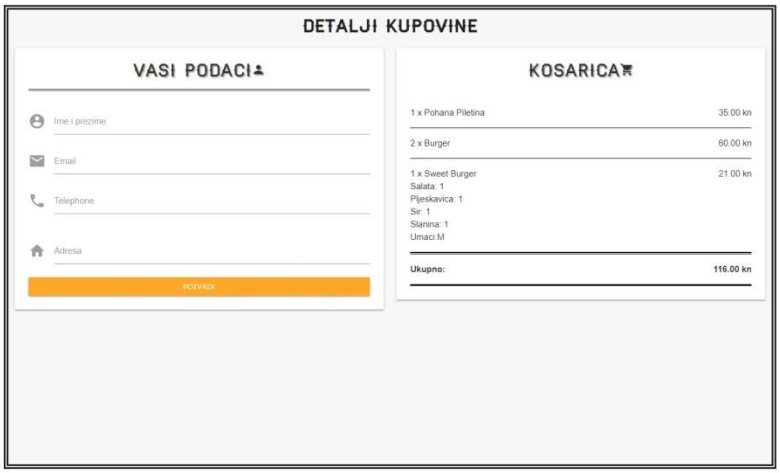
nasljeđuje *React.Component* klasu jer je potrebna *lifecycle* metoda *componentWillMount()* čiji je kod vidljiv ispod.

```
componentWillMount() {
  if (this.props.location.state === undefined) {
    this.props.history.push("/menu");
  }
}
```

componentWillMount() metoda poziva se prije *render()* metode što znači da komponenta zahtjeva izvršavanje određenih operacija prije nego bude prikazana na sučelju. U ovom slučaju provjeravamo jesmo li putem *Link* komponente proslijedili stanje košarice. Ako košarica nije ispunjena u korisničkom sučelju, te je kliknut gumb za nastavak kupovine, korisnik će biti vraćen na glavno korisničko sučelje aplikacije. To znači da će URL biti promijenjen u onaj koji prikazuje glavno sučelje. Pojednostavljeno, ukoliko korisnik ručno unese URL za *Checkout* komponentu, biti će vraćen na glavno korisničko sučelje jer je košarica prazna.

Checkout komponenta koristi *state* objekt za upravljanje poljima forme. Kada korisnik popuni sva obavezna polja vidljiva na Slici 3.8., i klikom na gumb potvrdi narudžbu, poziva se metoda *onSubmit()* vidljiva u kodu niže koja kreira objekt i u njega sprema sve podatke vezane uz narudžbu kao što su vrijeme narudžbe, podatci o korisniku, popis naručenih obroka, te svojstvo „*shipped*“ kojim manipulira admin u odjeljku za narudžbe. Nakon toga, poziva se *add()* metoda uvezena iz *react-redux-firebase* biblioteke koja u kolekciju „*orders*“ u Firebase bazi podataka dodaje novu narudžbu. Kada je narudžba uspješno pohranjena, poziva se metoda *onOpenModal()* koja korisniku daje do znanja da je narudžba zaprimljena. Isto tako, poziva se i ugrađena *setTimeout()* metoda koja korisnika vraća na početnu stranicu korisničkog sučelja dvije sekunde nakon što je poruka prikazana na zaslonu.

Svaka narudžba vidljiva je u odjeljku za narudžbe administracijskog sučelja.



Slika 3.8. Detalji kupovine.

```
onSubmit = e => {
  e.preventDefault();
  const { firestore } = this.props;
  const { totalPrice, orders } = this.props.location.state;
  const { name, email, tel, address } = this.state;

  let date = new Date();

  const orderData = {
    totalPrice,
    orders,
    contact: { name, email, tel, address },
    orderTime: date.toUTCString(),
    shipped: false
  };

  firestore.add({ collection: "orders" }, orderData).then(() => {
    this.onOpenModal();
    setTimeout(() => this.props.history.push("/menu"), 2000);
  });
};
```

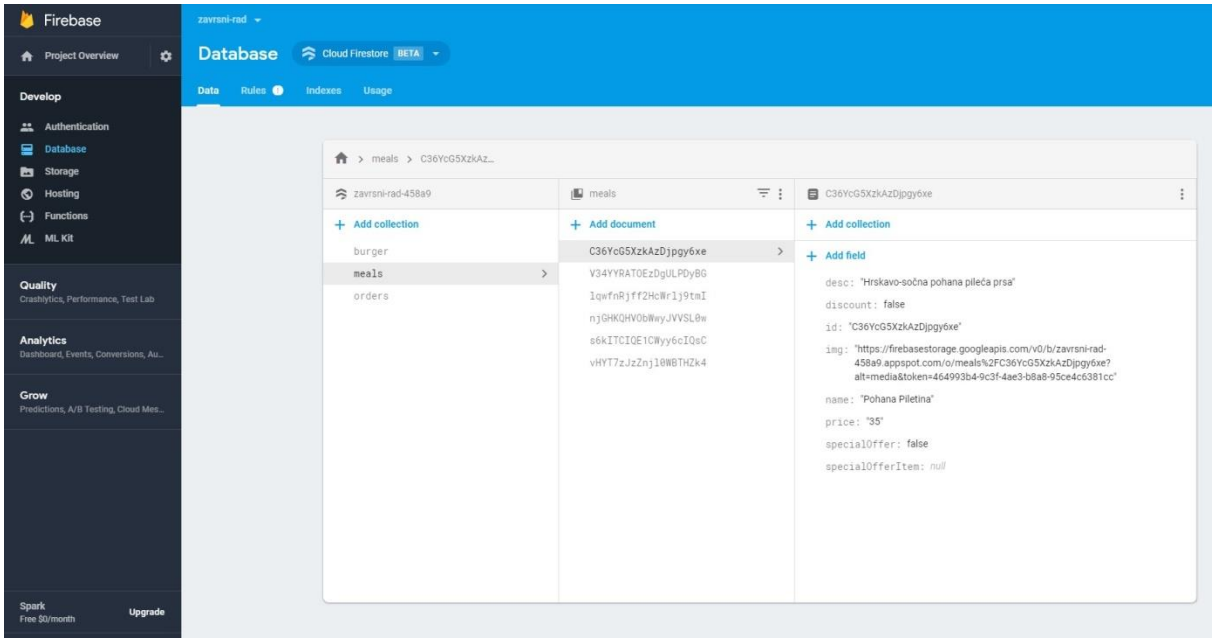
3.3. Testiranje aplikacije

Za testiranje aplikacije korišten je Google Chrome web preglednik, *Google Dev Tools* dodatci *React* i *Redux Dev Tools*.

React Dev Tools je dodatak za *Chrome DevTools* za besplatnu React JS biblioteku. On omogućava pregled svih React komponenata koje su učitane u stranicu. Odabirom pojedine komponente moguće je vidjeti trenutno stanje *state* i *props* objekta. Isto tako, moguće je i mijenjati te objekte i pratiti promjene na sučelju.

3.4. Firebase sučelje

Na Slici 3.9. vidljivo je Firebase sučelje i kolekcije podataka koje se koriste u ovoj aplikaciji.



Slika 3.9. Firebase sučelje.

4. ZAKLJUČAK

Zbog ubrzanog načina života, ljudi sve manje vremena provode u vlastitim kuhinjama i odlučuju se na dostavu hrane. Iz tog razloga se pojavila potreba pojednostaviti način narudžbe brze hrane. U moru sličnih aplikacija za naručivanje hrane, ova aplikacija se izdvaja i posebna je po tome što korisnik osim izbora gotovih jela, tj. menija, ima i mogućnost sastavljanja sendviča po vlastitoj želji u onim količinama koje želi. Naručivanje obroka je olakšano putem interaktivnog i jednostavnog sučelja, što je također jedna od prednosti ove aplikacije. Osim što je korisniku olakšano naručivanje i višestruko uštađeno vrijeme, tako se i vlasniku restorana olakšava rad jer lakše može manipulirati obrocima, uređivati menije, dodavati fotografije, pratiti statistiku narudžbi i slično.

Među mnogo JavaScript biblioteka koje se koriste za izradu aplikacija, potrebno je na temelju vlastitih preferencija prilikom izrade aplikacije odabrati najbolju jer svaka ima svoje prednosti i nedostatke. Za ovu aplikaciju izabrana je React JS biblioteka zbog svoje jednostavnosti i mogućnosti izrade aplikacije sastavljene od ponovno upotrebljivih cjelina. Unutar koda će u budućnosti biti moguće lako snalaženje i dodavanje promjena.

LITERATURA

[1] https://developer.mozilla.org/bm/docs/Web/JavaScript (posjećeno: 15. lipnja 2018.)

[2] https://hr.wikipedia.org/wiki/HTML (posjećeno: 15. lipnja 2018.)

[3] https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/HTML_basics (posjećeno: 15. lipnja 2018.)

[4] https://www.popwebdesign.net/sta_je_css.html (posjećeno: 15. lipnja 2018.)

[5] https://hackernoon.com/introduction-to-firebase-218a23186cd7 (posjećeno: 17.rujna 2018.)

[6] https://unsplash.com/ (posjećeno: 25. kolovoza 2018.)

[7] https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi (posjećeno: 25. kolovoza 2018.)

SAŽETAK

Cilj ovog rada bio je izraditi web aplikaciju za vođenje restorana brze prehrane koja će olakšati vlasniku restorana upravljanje narudžbama, te omogućiti korisniku jednostavno i brzo naručivanje obroka. Aplikacija ima interaktivno i moderno sučelje koje pruža jedinstveno iskustvo. U teorijskom dijelu obuhvaćen je opis JavaScript biblioteke React JS koji je glavni alat za izradu ove aplikacije. Aplikacija se sastoji od dva dijela, administratorskog i korisničkog. Administratorski dio služi za dodavanje, upravljanje i uređivanje obroka. Na kraju, administratorski dio služi i za vođenje statistike učestalosti naručivanja pojedinih obroka i praćenja narudžbi. Korisnički dio nudi jednostavno sučelje za odabir prethodno dodanih obroka, kao i za izradu vlastitog sendviča uz onu količinu sastojaka koju korisnik želi. Nakon odabira željenih obroka, za potvrdu narudžbe potrebno je upisati osobne podatke.

Ključne riječi: JavaScript, React JS, Firebase, korisnik, admin, obrok, narudžba

ABSTRACT

Web application for managing fast food restaurant

The goal of this paper was to create a web application for managing fast food restaurant that will make it easier for the restaurant owner to manage orders and allow the user to easily and quickly order meals. The application has an interactive and modern interface that provides an unique experience. The theoretical part descibes the JavaScript library React JS which is the main tool for creating this application. The application consists of two parts, administrator and user. The admin part serves to add, manage and edit the meal. Finally, the administrative part also serves to manage the frequency of ordering individual meals and order tracking. The user section offers a simple interface for selecting previously added meals as well as for making your own sandwich with the amout of ingredients that user wants. After selecting the desired food items, you must enter your order confirmation infromation.

Keywords: JavaScript, React JS, Firebase, user, admin, meal, order

ŽIVOTOPIS

Ante Tolušić je rođen 31.srpnja 1995. godine u Đakovu, Hrvatska. Ondje je završio osnovnoškolsko i srednjoškolsko obrazovanje. Srednju školu je završio 2014. u Srednjoj strukovnoj školi Antuna Horvata, smjer računalni tehničar za strojarstvo. Stručni studij informatike na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku upisuje 2014. godine, koji i dalje pohađa. Posjeduje vještine stečene srednjoškolskim obrazovanjem u dizajniranju i crtanju pomoću CAD alata kao što je AutoCAD. Samostalnim radom usvojio je mnoga znanja u području web developmenta. Trenutno je zaposlen kao Junior developer u zagrebačkoj tvrtci Maidea.

PRILOZI

https://tole9.github.io/react-restaurant-management/#/

https://github.com/tole9/react-restaurant-management